# Deep Learning for Edge AI: SqueezeNet CNN Training on Distributed ARM-based Clusters

Dimitrios Papakyriakou
PhD Candidate
Department of Electronic Engineering
Hellenic Mediterranean University
Crete, Greece

Ioannis S. Barbounakis
Assistant Professor
Department of Electronic Engineering
Hellenic Mediterranean University
Crete, Greece

## ABSTRACT
The increasing demand for lightweight and energy-efficient deep learning models at the edge has fueled interest in training convolutional neural networks (CNNs) directly on ARM-based CPU clusters. This study examines the feasibility and performance constraints of distributed training for the compact SqueezeNet v1.1 architecture, implemented using an MPI-based parallel framework on a Beowulf cluster composed of Raspberry Pi devices.

Experimental evaluation across up to 24 Raspberry Pi nodes (48 MPI processes) reveals a sharp trade-off between training acceleration and model generalization. While wall-clock training time improves by over (11×) under increased parallelism, test accuracy deteriorates significantly, collapsing to chance-level performance (≈10%) as data partitions per process become excessively small. This behavior highlights a statistical scaling limit, beyond which computational gains are offset by learning inefficiency. The findings are consistent with the statistical bottlenecks identified by Shallue et al. (2019) [11], extending their observations from large-scale GPU/CPU systems to energy-constrained ARM-based edge clusters.

These findings underscore the importance of balanced task decomposition in CPU-bound environments and contribute new insights into the complex interplay between model compactness, data sparsity, and parallel training efficiency in edge-AI systems. This framework also provides a viable low-power platform for real-time SNN research on edge devices.

## Keywords
SqueezeNet, Distributed Deep Learning, Edge Computing, Raspberry Pi Cluster, Beowulf Cluster, ARM Architecture, MPI (Message Passing Interface), Low-Power AI, Strong Scaling, Model Generalization, Statistical Scaling Limit.

## 1. INTRODUCTION
The rapid proliferation of artificial intelligence (AI) applications at the network edge—ranging from autonomous sensing systems to low-power surveillance and smart IoT endpoints—has accelerated the demand for resource-efficient deep learning (DL) models deployable on embedded hardware platforms [1], [2]. While modern convolutional neural networks (CNNs) achieve state-of-the-art accuracy across a variety of computer vision tasks, their deployment on lightweight hardware remains challenging due to memory, energy, and compute limitations [3].

Recent developments in model compression and architecture optimization have led to the emergence of compact CNN variants, such as *MobileNet* and *SqueezeNet*, which offer a favorable trade-off between inference speed and accuracy [4], [5]. These models have shown promising results for edge inference, yet the majority of existing studies rely on pre-trained models, limiting the scope of on-device adaptation and learning. In contrast, performing model training directly on edge hardware—particularly in a distributed fashion—remains largely unexplored due to the stringent constraints of low-power CPUs and lack of GPU acceleration [6], [7].

This work addresses the gap by investigating the feasibility of parallel CNN training using *SqueezeNet v1.1* on ARM-based edge clusters. Leveraging a message-passing interface (MPI) strategy across multiple Raspberry Pi devices, the analysis focuses on evaluating the trade-offs between training throughput, communication overhead, and statistical efficiency as the number of processes increases. Particular attention is given to the interaction between parallelism and model generalization, exposing a regime where increased scalability leads to diminishing learning returns.

By extending the frontier of embedded deep learning from inference-only systems toward scalable training architectures, this study examines *SqueezeNet v1.1* as a candidate for energy- and memory-efficient CNN deployment in clustered edge environments. Implementing MPI-based distributed training on ARM-based devices reveals both the strengths and the practical limits of this approach, highlighting the trade-offs between computational scalability, statistical efficiency, and generalization.

The experimental platform, illustrated in "Figure 1",[8], [9], is built on the Raspberry Pi 4 Model B with 8 GB LPDDR4 RAM and a 64-bit quad-core ARMv8 Cortex-A72 CPU at 1.5 GHz, chosen for its low cost, accessibility, and suitability for high-performance cluster assembly at the edge. This hardware foundation provides a controlled and repeatable basis for evaluating parallel processing and distributed deep learning workloads, enabling insights that contribute to the design of future edge-AI systems less dependent on cloud infrastructure.

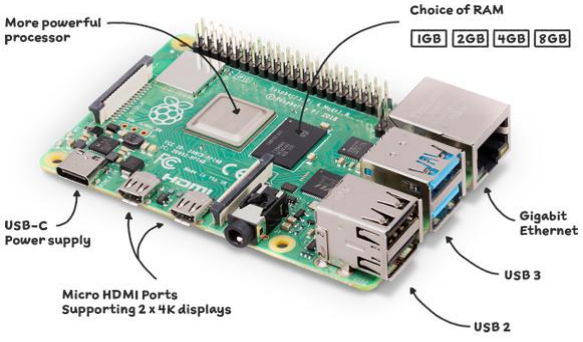


**Figure 1: Raspberry Pi 4 Model B unit used as the basic node in the 24-node ARM-based Beowulf cluster.**

# 2. SYSTEM DESCRIPTION

## 2.1 Hardware Equipment

The computational platform for this study is a cost-effective yet capable Beowulf-style cluster built from 24 Raspberry Pi 4 Model B units, each equipped with 8 GB LPDDR4 RAM. One board is configured as the master node, responsible for resource allocation and process orchestration, while the remaining 23 nodes serve as MPI-coordinated workers executing parallel training tasks. The physical layout "Figure 2" consists of four vertical stacks containing six boards each, a format that offers both space efficiency and clean, maintainable cabling.

High-speed inter-node communication is provided by TP-Link TL-SG1024D unmanaged Gigabit Ethernet switches, delivering 1 Gbps full-duplex bandwidth per link. This topology ensures consistent, low-latency data exchange between nodes, effectively replicating the communication characteristics of a traditional high-performance computing (HPC) environment within an ARM-based embedded system.

Stable and reliable power delivery is maintained through two industrial-grade switch-mode power supplies, each rated at 60 A / 5 V and precisely tuned to 5.80 V to offset voltage drops over extended cabling. This adjustment safeguards node stability during sustained, high-load parallel operations.

For storage, the *master node* hosts a 1 TB Samsung 980 PCIe 3.0 NVMe SSD to provide high-throughput access for dataset management and orchestration tasks. Each worker node is fitted with a 256 GB Patriot P300 NVMe M.2 SSD, ensuring fast local I/O to support seamless data streaming during training. This configuration supplies adequate storage bandwidth for large-scale datasets and for maintaining intermediate model checkpoints throughout distributed learning.

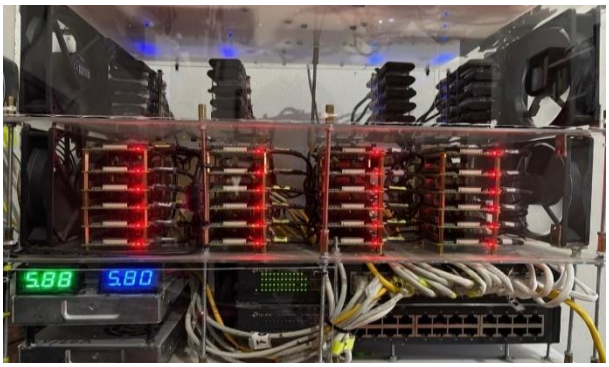


**Figure 2: 24-node Raspberry Pi 4B (8 GB) Beowulf cluster architecture for distributed SqueezeNet training.**

## 2.2 Software Environment and Toolchain

The software environment utilized in this study was carefully designed to support distributed training of lightweight convolutional neural networks on ARM-based systems. All nodes operated under Raspberry Pi OS 64-bit Lite (Debian-based), with Python 3.11.5 running within a shared virtual environment mounted via NFS, ensuring a uniform software stack and consistent execution across the entire cluster.

Inter-process communication was managed using MPICH v4.2.0, while the distributed training logic was implemented with the mpi4py library (v3.1.6), providing a Pythonic interface to the underlying MPI runtime.

In addition to core scientific libraries - including *NumPy (v1.26.4)*, *SciPy (v1.13.0), scikit-learn (v1.4.2),* and *psutil (v7.0.0)* - the software stack incorporated deep learning-specific frameworks required for CNN construction, training, and evaluation. Specifically, *TensorFlow v2.15.0*, with integrated *Keras APIs*, was employed to support the modular architecture of *SqueezeNet* and to facilitate GPU-independent model training and evaluation.

All dependencies were compiled for compatibility with the ARMv8-A architecture, and deployed uniformly to ensure deterministic behavior and eliminate version drift. To support efficient parallelism, environmental variables such as LD_LIBRARY_PATH and UCX_TLS were explicitly synchronized across all nodes, and passwordless SSH was configured to allow seamless coordination during training execution.

The entire training process was orchestrated using mpiexec, with explicit machine file definitions and CPU core binding to optimize resource allocation and minimize scheduling variability. All operations were executed from within the shared virtual environment, ensuring that experimental reproducibility and consistency were maintained throughout the distributed training workflow.

This configuration was tailored to the constraints and capabilities of *SqueezeNet*, allowing for scalable training without GPU support, and demonstrating the viability of lightweight CNNs in fully distributed ARM-based edge environments. This environment replicates many practices from large-scale HPC clusters, adapted to resource-constrained ARM nodes.

## 2.3 Design

The architectural and experimental framework for distributed training is shown in "Figure 2" and "Figure 3", illustrating both the physical deployment and the logical data-parallel workflow of the Raspberry Pi 4B Beowulf cluster. The system consists of 24 ARM-based nodes, each with 8 GB of RAM and interconnected via Gigabit Ethernet, forming a cost-effective yet representative platform for investigating distributed deep learning under resource constraints.

This work focuses on the distributed training of *SqueezeNet v1.1*, a compact convolutional neural network that achieves AlexNet-level accuracy while maintaining a model size below (0.5 MB). In contrast to *MobileNet*, which leverages depthwise separable convolutions to reduce computation, *SqueezeNet v1.1* employs a distinctive modular design centered on "fire modules"—comprising squeeze ( $1 \times 1$ convolution) and expand ($1 \times 1$ and $3 \times 3$ convolution) layers. This architecture delivers aggressive parameter reduction without significant loss in accuracy, making it highly suitable for CPU-only embedded devices such as the Raspberry Pi.

The ultra-small footprint and low computational demand of *SqueezeNet v1.1* make it an ideal candidate for exploring fully distributed training in scenarios characterized by limited memory, absence of GPU acceleration, and moderate communication latency. These constraints closely mirror those encountered in practical edge computing deployments, including decentralized applications in robotics, environmental sensing, and IoT-driven event detection.

Given the platform's lack of dedicated accelerators—such as GPUs, Advanced Vector Extensions (AVX) (Single Instruction, Multiple Data) SIMD extensions, or high-throughput Direct Memory Access (DMA)—the entire neural network workload is executed solely on quad-core ARM Cortex-A72 CPUs. To accommodate these limitations, the

experimental protocol adopts:

- Smaller per-node batch sizes to conserve memory and reduce compute overhead,
- A fixed, moderate number of training epochs (e.g., 10) to maintain tractable runtimes,
- Synchronous data-parallel training using MPI, wherein each process operates on a partitioned data shard and participates in collective gradient aggregation.

While the objective is not to attain state-of-the-art accuracy or speed, this design enables:

- Comprehensive benchmarking of strong scaling performance as the number of processes increases (from 2 to 48),
- Accurate measurement of parallel efficiency for a compact CNN under distributed execution,
- Practical demonstration of the feasibility of on-device learning on embedded, multi-node ARM clusters.

This methodology establishes a reproducible framework for assessing distributed CNN training under realistic edge constraints, offering quantifiable evidence of the trade-offs between communication overhead and statistical learning performance.

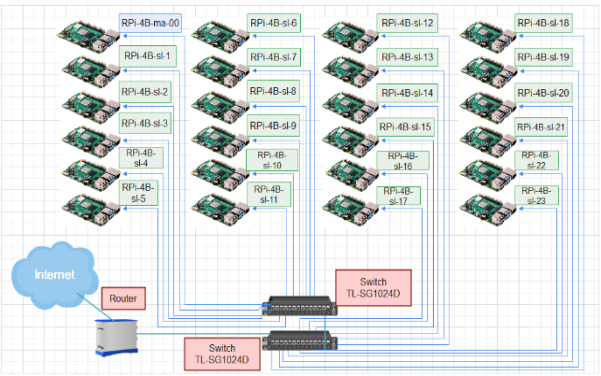


**Figure 3: MPI communication architecture of the 24-node Raspberry Pi 4B cluster.**

## 2.4 Theoretical Background: SqueezeNet CNNs and Dataset Selection

Convolutional Neural Networks (CNNs) constitute a foundational class of deep learning models widely used in image recognition tasks, owing to their ability to extract multiscale hierarchical features from visual data [7]. Among the numerous lightweight CNN architectures, *SqueezeNet* has emerged as a notable design, achieving AlexNet-level classification accuracy with $50\times$ fewer parameters and a model size of less than (0.5MB) when quantized [8]. This efficiency is achieved through a unique architectural innovation: the "*fire module*", which replaces standard convolutional layers with a two-stage module composed of squeeze ($1 \times 1$ convolution) and expand ($1 \times 1$ and $3 \times 3$ convolution) layers

The *SqueezeNet* architecture eliminates the need for computationally intensive fully connected layers and instead utilizes global average pooling in its final stages, further reducing parameter count while maintaining strong representational power. Its highly compact form and modular structure make it particularly suitable for deployment in memory- and energy-constrained environments -, such as ARM-based embedded platforms.

For this investigation, *SqueezeNet* was selected for training on the CIFAR-10 dataset, a commonly used image classification benchmark consisting of 60,000 color images at ($32 \times 32$) resolution, evenly distributed across 10 classes [9]. The dataset includes 50,000 training images and 10,000 test images, providing a balanced and computationally tractable workload for evaluating the distributed training behavior of compact CNNs under hardware constraints.

Preliminary testing indicated that CIFAR-10 aligns well with the architectural strengths of *SqueezeNet*, enabling successful training without exceeding the memory limits or thermal thresholds of the Raspberry Pi 4B cluster. In contrast, larger or more complex datasets such as CIFAR-100 - which increases the output class count and dense layer complexity tenfold - introduced significant memory pressure and degraded synchronization efficiency, underscoring the importance of dataset-model alignment when evaluating distributed training on constrained hardware.

Given these constraints and objectives, CIFAR-10 was selected as the benchmark dataset for this study to ensure stability during execution, while still offering sufficient complexity for analyzing model convergence and scaling performance. The dataset's modest size complements the efficient design of SqueezeNet, making it an ideal choice for parallel execution and evaluation on low-power embedded clusters.

# 3. METHODOLOGY

## 3.1 System Configuration and Experimental Context

The experimental evaluation was conducted on a custom-built Beowulf-style cluster composed of 24 Raspberry Pi 4 Model B nodes, each featuring a quad-core ARM Cortex-A72 CPU and 8GB of LPDDR4 RAM. The nodes are interconnected using unmanaged Gigabit Ethernet switches, providing a full-duplex communication channel with 1 Gbps per link, thereby enabling low-latency inter-node messaging suitable for MPI-based coordination.

This embedded, low-power cluster architecture presents a distinct set of computational constraints, including limited memory, absence of GPU or AVX acceleration, and CPU-bound processing. These limitations directly influence the design and tuning of the training pipeline, particularly in terms of model selection, dataset size, and parallelization strategy.

To accommodate these constraints, the study employs the *SqueezeNet* convolutional neural network - a lightweight deep learning architecture specifically designed to minimize model size while retaining high classification accuracy-. Unlike more computationally intensive CNNs, SqueezeNet utilizes *fire modules* to reduce the number of parameters, replacing traditional convolutional layers with combinations of squeeze $1 \times 1$ and expand ($1 \times 1$ and $3 \times 3$) convolutions. This compact structure makes the model ideally suited for training in CPU-only, memory-constrained environments such as the Raspberry Pi 4B platform

The CIFAR-10 dataset is selected as the training and evaluation benchmark. It consists of 60,000 RGB images ( $32 \times 32$ pixels), evenly distributed across 10 object categories, with 50,000 samples for training and 10,000 for testing. Its moderate size aligns well with the available memory and I/O capabilities

of the cluster, permitting full dataset preloading into memory to avoid paging or I/O bottlenecks. In accordance with CIFAR-10 dimensions, the input tensor for SqueezeNet is modified to accept input shape $(32 \times 32 \times 3)$ instead of its default $227 \times 227 \times 3$), and all internal layers are adapted to ensure shape compatibility and effective training at lower resolution.

All cluster nodes run Raspberry Pi OS 64-bit Lite, and training is conducted within a shared *Python 3.11.5* virtual environment, mounted across nodes via NFS for consistency. The distributed training framework is built upon *TensorFlow v2.15.0*, leveraging the *Keras 2.x API* for model construction, optimization, and evaluation. Notably, compatibility adjustments were applied to the open-source *keras-squeezenet* module to ensure full support under the *TensorFlow 2.x* backend, replacing legacy Keras imports and resolving API deprecations introduced in recent versions.

Inter-process communication is implemented using the MPI for Python (*mpi4py) library v3.1.6* over the *MPICH v4.2.0* backend. All software packages are compiled and installed with ARMv8-A architecture compatibility, and deployed uniformly across the cluster to ensure consistent behavior. MPI processes are launched using mpiexec, with machinefile definitions and CPU core binding for improved locality, control of process placement, and reproducibility of results.

This configuration supports the investigation of strong scaling behaviour in lightweight CNN training, with particular focus on training time, convergence speed, and efficiency metrics across increasing numbers of MPI processes - ranging from (2 to 48), spanning (1 to 24) physical nodes.

## 3.2 Distributed Training Strategy

### *3.2.1 Data Parallelism Approach*

This study adopts a data-parallel distributed training strategy, leveraging Message Passing Interface (MPI) to coordinate multiple processes across the cluster. Each MPI process ("rank") is assigned a unique, disjoint shard of the CIFAR-10 training dataset. The training pipeline is configured to run two ranks per Raspberry Pi 4B node, optimizing both CPU core utilization and memory headroom on the ARM Cortex-A72 architecture. By ensuring that each rank processes a distinct chunk of data per epoch, the approach guarantees balanced workload distribution and reproducibility across all experimental runs.

### *3.2.2 Deterministic Strided Sharding of CIFAR-10*

To maintain reproducible and uniform data splits during distributed *SqueezeNet v1.1* training, the 50,000 CIFAR-10 training samples are partitioned using a deterministic strided indexing scheme.

Let ( $N$ ) denote the total number of MPI ranks in the experiment and ( $r$ ) the rank identifier ($r \in \{0, 1, \dots, N-1\}$ ). For each epoch, the global training index set

( $I = 0,1,2, \dots ,49999$), is divided so that rank ( $r$ ) processes the subset:

$I_r = \{i \in I \mid i \bmod N = r\} = \{r, r+N, r+2N, \dots\}$.

This ensures non-overlapping data partitions, balanced sample counts ($\approx \frac{50{,}000}{N}$ $per\ rank$) and complete reproducibility across runs.

By construction, every sample is assigned to exactly one rank in each epoch, eliminating duplication and statistical bias that could arise from random shuffling. After local training on its assigned shard, each rank participates in global model synchronization, and the root rank performs test set evaluation using the fully synchronized model parameters.

### *3.2.3 Synchronous SGD via MPI Collectives*

Within each epoch, every rank performs standard forward and backward passes on its local data shard. At the end of the epoch, synchronous stochastic gradient descent (SGD) is emulated via MPI collectives: all ranks transmit their current model weight tensors to the root process using *comm.gather()*. The root averages these tensors across ranks, then broadcasts the aggregated (synchronized) weights back to all ranks using *comm.bcast()*. This collective pattern is functionally equivalent to *Allreduce*, but was selected for transparency and enhanced logging. Test evaluations are executed by the root rank using the fully synchronized model, ensuring that the final metrics reflect the converged global weights.

### *3.2.4 Empirical Batch Size Profiling and System Sweeps*

Batch size is the primary factor influencing per-rank memory usage and CPU saturation in distributed CNN training. A systematic "*fast sweep*" protocol was implemented—batch sizes (*8, 16, 24, 32 images/node*) were benchmarked in isolation on a single RPi (2 ranks/node), with additional spot-checks on further nodes to rule out hardware variation. For each sweep: *epoch times*, *peak process memory* (RSS via psutil), *node-level RAM/swap usage* (free -m, vmstat), *CPU utilization* (mpstat), and *OOM events* (dmesg) were logged for validation. Results indicated that batch-per-node = 24 achieves the optimal trade-off: highest throughput, consistent CPU saturation (>107%), and safe memory usage (< 660 MB/rank). Larger batches induced minor cache pressure and slower epochs; smaller batches under-utilized CPU resources. Cross-node replicates confirmed stability "Figure 4", "Figure 5", "Table 1".

### *3.2.5 Benchmarking Hyperparameters and Reproducibility*

To ensure consistency across all scaling experiments, benchmarking hyperparameters were fixed as follows:

- *Epochs*: 10 (upper bound before RAM or thermal instability on sustained workloads)
- *Batch size*: 24 images/node (12 per rank with 2 ranks/node)
- *Learning rate*: 0.0005 (chosen for stable convergence under small effective batch)
- *Initialization & Codebase*: All runs used identical random seeds, an invariant training script, and uniform logging format for reproducibility.
- *Logging*: Per-rank logs included epoch loss/accuracy, wall-clock times, and peak RSS. All data was stored in CSV for downstream analysis and verification.

Batch size (24) was empirically determined from throughput/memory/CPU sweeps as the ideal setting, balancing compute density and system safety, and enabling linear scaling with additional nodes. Peak RSS remained well below RPi limits, with each rank saturating its core allocation, and data-parallel scaling was robust under increasing cluster size.

### 3.2.6 Fixed benchmarking hyperparameters in one RPi (pi@rpi4B-ma-00)

To empirically determine the optimal training configuration for SqueezeNet v1.1 on ARM-based edge hardware, a systematic sweep of primary hyperparameters was performed on a single Raspberry Pi 4B node (8 GB RAM, quad-core ARM Cortex-A72), running two MPI ranks per node. This sweep evaluated batch sizes of (8, 16, 24, and 32) images per node (i.e., 4, 8, 12, and 16 images per rank, respectively) under identical runtime conditions. For each configuration, we logged median epoch time, mean step time, global throughput (images/sec), peak per-process resident set size (RSS), and epoch-averaged CPU usage.

Empirical benchmarking revealed that *throughput* increases markedly from batch ($8 \rightarrow 16 \rightarrow 24$), peaking at (19.92 img/s) for batch 24/node, before dropping to (18.58 img/s) at batch 32/node. This decline at the largest batch size is attributed to the ARM Cortex-A72's memory subsystem limits: the larger working set of activations/gradients at batch 32 begins to exceed the effective capacity of the L2 cache, resulting in increased DRAM traffic, higher cache miss rates, and longer step times (mean step rises from 1.205 s at 24 to 1.723 s at 32) "Table 1".

Peak RSS also trends upward at batch 32 (655.7 MB/rank) compared to ($\leq$ 638 MB/rank) for smaller batches, indicating greater memory pressure.

Across all runs, *CPU utilization* remained high (> 107 % per process), confirming full use of allocated cores, and no OOM or swap events occurred. Based on this profiling, batch-per-node = 24 was fixed for all subsequent scaling experiments as it delivers the highest throughput, maintains comfortable memory headroom, and avoids the bandwidth/cache penalties observed at larger batch sizes.

```
(venv) pi@rpi4B-ma-00:~/cloud $ mpiexec -n 2 python mpi_squeezenet_cifar_sweep_v
1_fast.py \--data-root ~/.keras/datasets \--batch-per-node 8 --ranks-per-node 2
\--epochs 1 --repeats 1 --lr 5e-4 \--num-workers 0 --seed 42 \--input-size 128 -
-channels-last \--max-steps 200

==== MPI SqueezeNet v1.1 CIFAR-10 (FAST SWEEP MODE) ====
Host (rank0): rpi4B-ma-00
World size: 2  |  Ranks/node: 2  |  Nodes: 1
Batch/node: 8  |  Batch/rank: 4  |  Global batch: 8
Epochs: 1  |  Repeats: 1  |  LR: 0.0005
Input size: 128  |  channels_last: True  |  width_mult(ignored): 1.0
Max steps/epoch: 200  (effective samples/epoch: 1600)
Train samples (total): 50000  |  Steps/epoch used: 200
--- Results ---
Avg epoch time (max across ranks): 91.633 s
Mean step time: 0.458 s/step
Global throughput (effective): 17.46 img/s
Peak RSS (max across ranks): 638.1 MB
CPU% System  (epoch-avg): local=54.7  mean_ranks=54.7  max_ranks=54.7
CPU% Process (epoch-avg): local=108.0 mean_ranks=108.2 max_ranks=108.4
================================================

(venv) pi@rpi4B-ma-00:~/cloud $
(venv) pi@rpi4B-ma-00:~/cloud $
(venv) pi@rpi4B-ma-00:~/cloud $ mpiexec -n 2 python mpi_squeezenet_cifar_sweep_v
1_fast.py \--data-root ~/.keras/datasets \--batch-per-node 16 --ranks-per-node 2
 \--epochs 1 --repeats 1 --lr 5e-4 \--num-workers 0 --seed 42 \--input-size 128
--channels-last \--max-steps 200

==== MPI SqueezeNet v1.1 CIFAR-10 (FAST SWEEP MODE) ====
Host (rank0): rpi4B-ma-00
World size: 2  |  Ranks/node: 2  |  Nodes: 1
Batch/node: 16  |  Batch/rank: 8  |  Global batch: 16
Epochs: 1  |  Repeats: 1  |  LR: 0.0005
Input size: 128  |  channels_last: True  |  width_mult(ignored): 1.0
Max steps/epoch: 200  (effective samples/epoch: 3200)
Train samples (total): 50000  |  Steps/epoch used: 200
--- Results ---
Avg epoch time (max across ranks): 166.142 s
Mean step time: 0.831 s/step
Global throughput (effective): 19.26 img/s
Peak RSS (max across ranks): 637.7 MB
CPU% System  (epoch-avg): local=54.4  mean_ranks=54.4  max_ranks=54.4
CPU% Process (epoch-avg): local=107.7 mean_ranks=107.8 max_ranks=107.9
================================================
```

**Figure 4: Local training benchmark on a single RPi comparing batch sizes 8 and 16 for computational efficiency.**

```
(venv) pi@rpi4B-ma-00:~/cloud $ mpiexec -n 2 python mpi_squeezenet_cifar_sweep_v
1_fast.py \--data-root ~/.keras/datasets \--batch-per-node 24 --ranks-per-node 2
 \--epochs 1 --repeats 1 --lr 5e-4 \--num-workers 0 --seed 42 \--input-size 128
--channels-last \--max-steps 200

==== MPI SqueezeNet v1.1 CIFAR-10 (FAST SWEEP MODE) ====
Host (rank0): rpi4B-ma-00
World size: 2  |  Ranks/node: 2  |  Nodes: 1
Batch/node: 24  |  Batch/rank: 12  |  Global batch: 24
Epochs: 1  |  Repeats: 1  |  LR: 0.0005
Input size: 128  |  channels_last: True  |  width_mult(ignored): 1.0
Max steps/epoch: 200  (effective samples/epoch: 4800)
Train samples (total): 50000  |  Steps/epoch used: 200
--- Results ---
Avg epoch time (max across ranks): 241.010 s
Mean step time: 1.205 s/step
Global throughput (effective): 19.92 img/s
Peak RSS (max across ranks): 637.6 MB
CPU% System  (epoch-avg): local=54.2  mean_ranks=54.2  max_ranks=54.3
CPU% Process (epoch-avg): local=107.5 mean_ranks=107.5 max_ranks=107.6
================================================

(venv) pi@rpi4B-ma-00:~/cloud $ mpiexec -n 2 python mpi_squeezenet_cifar_sweep_v
1_fast.py \--data-root ~/.keras/datasets \--batch-per-node 32 --ranks-per-node 2
 \--epochs 1 --repeats 1 --lr 5e-4 \--num-workers 0 --seed 42 \--input-size 128
--channels-last \--max-steps 200

==== MPI SqueezeNet v1.1 CIFAR-10 (FAST SWEEP MODE) ====
Host (rank0): rpi4B-ma-00
World size: 2  |  Ranks/node: 2  |  Nodes: 1
Batch/node: 32  |  Batch/rank: 16  |  Global batch: 32
Epochs: 1  |  Repeats: 1  |  LR: 0.0005
Input size: 128  |  channels_last: True  |  width_mult(ignored): 1.0
Max steps/epoch: 200  (effective samples/epoch: 6400)
Train samples (total): 50000  |  Steps/epoch used: 200
--- Results ---
Avg epoch time (max across ranks): 344.515 s
Mean step time: 1.723 s/step
Global throughput (effective): 18.58 img/s
Peak RSS (max across ranks): 655.7 MB
CPU% System  (epoch-avg): local=56.1  mean_ranks=56.1  max_ranks=56.1
CPU% Process (epoch-avg): local=111.1 mean_ranks=111.2 max_ranks=111.3
================================================
```

**Figure 5: Local training benchmark on a single RPi comparing batch sizes 24 and 32 for computational efficiency.**

**Table 1. MPI SqueezeNet CIFAR-10, batch (Images) size benchmarking -best option survey-.**

| Batch /node | Batch /rank | Avg. epoch time (sec) | Mean step (sec) | Global through -put (img/s) | Peak RSS (MB) | Avg. Proc -ess CPU (%) |
|---|---|---|---|---|---|---|
| 8 | 4 | 91.63 | 0.458 | 17.46 | 638.1 | 108.2 |
| 16 | 8 | 166.14 | 0.831 | 19.26 | 637.7 | 107.8 |
| 24 | 12 | 241.01 | 1.205 | 19.92 | 637.6 | 107.5 |
| 32 | 16 | 344.52 | 1.723 | 18.58 | 655.7 | 111.2 |

*Note 1*:

*Reported process CPU utilization values >100% per MPI rank are expected in multi-core environments, as each rank may utilize more than one core through internal multi-threading (e.g., BLAS, data preprocessing). With two MPI ranks per node on a quad-core ARM CPU, this reflects efficient multi-threaded utilization and confirms that both ranks are fully leveraging available compute resources during distributed training "Table 1".*

*Note 2*:

*RSS is a direct indicator of the memory "footprint" of the training job at runtime, which shows how much RAM the deep learning code is consuming per process (MPI rank) without counting swapped or inactive pages. RSS is the live amount of physical RAM a process occupies; it's a key safety and efficiency metric in distributed deep learning benchmarking and hardware profiling. "Peak RSS" is the highest RSS observed for a rank throughout the training run "Table 1".*

## 3.3 Strong Scaling Training Configuration and Execution Flow

This study adopts a strong scaling experimental design to evaluate the distributed training performance of SqueezeNet on a CPU-bound, ARM-based embedded cluster. Strong scaling is critical in the context of edge AI because distributed training on resource-constrained hardware demands not only model compactness but also efficient parallelism. Unlike weak scaling, where data volume increases with compute resources, strong scaling directly reflects how well a fixed workload can be accelerated by adding more nodes - an essential metric for real-time, power-sensitive edge deployments-. In a strong scaling scenario, the total workload remains constant - specifically, the CIFAR-10 dataset and the SqueezeNet architecture - while the number of MPI processes (np) is gradually increased to assess parallel training behavior. The CIFAR-10 dataset was selected for its moderate size and compatibility with the memory and I/O constraints of the Raspberry Pi platform.

Experiments are executed with the following MPI process counts:

$$np \in \{2, 4, 8, 16, 24, 32, 40, 48\}$$

Given that each Raspberry Pi node executes two MPI processes, this corresponds to: $p = \frac{np}{2}$ Raspberry Pi nodes.

In the experiments:

- The overall dataset (CIFAR-10, 50,000 training samples, 10,000 test samples) and SqueezeNet model architecture remain unchanged for all trials.
- The number of MPI processes (np) is successively increased, while the same workload and data partitioning protocol is preserved.
- Each Raspberry Pi node executes two MPI processes (ranks), so the total process count np corresponds to np/2 physical Raspberry Pis.
- *Training Protocol*: Each MPI rank receives a non-overlapping data shard (via deterministic strided indexing) and performs local updates using synchronous, data-parallel SGD. Model gradients are aggregated via mpi4py collective operations (*gather* and *broadcast*), emulating centralized synchronous parameter updates. The training is run for a fixed 10 epochs—empirically chosen as the maximal stable setting before memory, swap, or thermal fluctuations arise on the ARM edge platform.
- *Experimental Controls:*

  All training runs use:

  - Fixed learning rate (0.0005) and uniform batch size per rank (chosen via single-node empirical sweep).
  - Identical random seed and weight initialization.
  - Shared codebase, containerized virtual environment, and NFS-based distribution to enforce bitwise reproducibility.
  - Explicit machinefile specification, core binding, and mpiexec process launching to guarantee locality and consistent scheduling.
  - Each configuration is repeated three times; results are reported as cross-run means to account for stochasticity and environmental noise.
- *Logged Metrics and Performance Indicators*: Each rank records its local loss and accuracy per epoch into structured (.csv) logs. Upon completion, these logs are aggregated to compute key performance indicators:

  - Mean Train Accuracy Across Ranks:

  $$MeanTraningAcc = \frac{1}{N}\sum_{i=1}^{N} Acc_i$$

  Where:

  $N = number\ of\ ranks \in \{2, 4, 8, 16, 24, 32, 40, 48\}$

  $Acc_i = final\ accuracy\ of\ rank\ (i)\ after\ epoch\ 10$

  - Total training time (as the maximum clock time over ranks)
  - Speedup S(np):

  $$S(np) = \frac{T_{base}}{T_{np}}$$

  Where (*Tbase)* is the runtime for the baseline case (np = 2)

  - Parallel Efficiency E(np):

  $$E(np) = \frac{S_{(np)}}{\frac{np}{2}} x\ 100\%$$

  *Note:*
  *Efficiency is reported relative to a baseline of np=2 MPI processes.*
- *Additional Considerations:*
  - Convergence dynamics are assessed using epoch-wise plots of training loss and accuracy.
  - Communication overhead is inferred from rising time variance and efficiency drops at high process counts (e.g., np ≥ 32).
  - All experiments are executed in thermal isolation, with passive cooling and controlled ambient conditions, to eliminate performance skew due to thermal throttling or network noise.
  - Each configuration is repeated for reproducibility, with averaged metrics used for all reported values.

This design allows for a detailed investigation into the scaling limits and distributed training feasibility of lightweight CNNs like *SqueezeNet* on low-power, decentralized edge infrastructures.

## 3.4 Strong Scaling Results and Analysis

To assess the strong scaling behavior of the SqueezeNet CNN on a distributed Raspberry Pi cluster, a series of training experiments were conducted using MPI-based data parallelism, varying the number of Raspberry Pi nodes and MPI processes. All configurations performed 10 epochs of training on a shared dataset with identical preprocessing, network architecture, and

hyperparameter settings. Key metrics collected include the mean training loss, mean training accuracy, total training time (maximum across ranks), speedup ($S_n$) relative to the baseline, and parallel efficiency ($E_n$).

- *Case 1: SqueezeNet-CNN_rpi-1_mpi-2*:

The baseline configuration was executed with *np = 2* MPI processes on a single Raspberry Pi 4B node for 10 training epochs. Each MPI process was bound to a dedicated core and accessed the dataset through a shared NFS mount. Although no inter-node communication was present, inter-process synchronization and gradient aggregation were still coordinated via *mpi4py*, introducing local network stack overhead and I/O contention on the shared storage.

The global wall-clock training time was (5003.00 sec) (measured as the maximum rank duration). The mean training accuracy across ranks reached (58.74 %), with a mean training loss of (1.1403). Final evaluation on the CIFAR-10 test set yielded (61.61 %) accuracy and (1.0779) test loss. These results establish that SqueezeNet v1.1 converges successfully within 10 epochs under CPU-only execution, even when restricted to a single embedded node.

This case defines the baseline reference point for all subsequent strong scaling experiments, with speedup defined as ($S_2 = 1$) and parallel efficiency as ($E_2 = 100$ %) *(baseline)*. While limited by single-node throughput and NFS-driven overheads, the configuration demonstrates that a compact CNN can achieve non-trivial generalization performance on CIFAR-10 in a fully ARM-based edge environment. It thereby provides the essential reference against which multi-node scaling behavior is evaluated "Figure 6", “Table 2”.

*Note*:

> *The close agreement between the final test loss (1.0779) and the mean training loss (1.1403) indicates that the model does not overfit under this configuration. Instead, the training process achieves a balanced fit where generalization to unseen data is consistent with training performance.*
>
> *In the context of deep learning, (loss) quantifies the discrepancy between the model’s predicted outputs and the true labels of the dataset. A lower loss value corresponds to fewer errors in prediction, whereas a high loss signals poor alignment with ground truth. Minimizing loss during training ensures that the model improves its predictive accuracy, and the similarity between training and test loss demonstrates the model’s ability to generalize beyond the data it was explicitly trained on*.

- *Case 2: SqueezeNet-CNN_rpi-2_mpi-4*:

The second configuration was executed with *np = 4* MPI processes distributed across two Raspberry Pi 4B nodes (*2 ranks per node*) for 10 training epochs. Each rank was bound to a dedicated CPU core, and dataset shards were allocated deterministically using the strided indexing scheme, resulting in 12,500 training samples per rank. Gradient aggregation and weight synchronization were managed synchronously via *mpi4py* collective communication, introducing the first instance of genuine inter-node message passing in the scaling experiments.

The global wall-clock training time was (2535.56 sec)**,** reflecting a near (2×) reduction relative to the single-node baseline. The mean training accuracy across ranks decreased to (43.45 %), with a mean training loss of (1.5139). Evaluation on the CIFAR-10 test set yielded (46.60 %) accuracy and (1.4356) test loss "Figure 7", “Table 2”.

Relative to (Case 1), this experiment achieved a clear runtime improvement while exhibiting reduced convergence quality. Speedup for np = 4 was ($S_4 \approx 1.97$), corresponding to a very high parallel efficiency of ($E_4 \approx 98.7$ %) under the np = 2 baseline definition. This indicates that doubling the number of nodes nearly halved training time, and the overhead of inter-node communication remained minimal at this scale. In other words, at (np = 4) the system retained excellent computational efficiency, and the observed accuracy drop was primarily due to statistical factors (smaller shard sizes per rank) rather than communication bottlenecks.

*Note*:

> *The divergence between mean training loss (1.5139) and test loss (1.4356) is relatively small, suggesting that the reduced accuracy arises not from overfitting but from statistical inefficiency in distributed gradient averaging at this scale [10], [11]. In deep learning, loss quantifies the penalty between predicted outputs and ground-truth labels. Here, the moderate loss values confirm that the model continues to learn effectively, though generalization accuracy diminishes compared to the baseline.*

- *Case 3: SqueezeNet-CNN_ rpi-1_mpi-2 to rpi-24_mpi-48 - Cluster-Wide Analysis and Observed Scaling Patterns:*

The cluster-wide configuration was executed with (*np = 48)* MPI processes distributed across 24 Raspberry Pi 4B nodes (*2 ranks per node*), representing the maximum scale of the experimental platform. Each rank received a deterministic strided shard of the CIFAR-10 dataset, corresponding to only (≈ 1,042) training samples per rank due to dataset partitioning across 48 processes. The global wall-clock training time decreased substantially to (432.11) seconds, marking a (11.6×) speedup compared to the single-node baseline (*Case 1*). However, mean training accuracy collapsed to (10.10 %), with a mean training loss of (2.3026**)**. Test accuracy mirrored this collapse at (10.00 %), with test loss also saturating at (2.3026). These values are near-random guess performance for CIFAR-10, indicating that the model failed to converge at scale. “Table 2”, "Figure 8", "Figure 9", "Figure 10", "Figure 11", "Figure 12".

While strong scaling produced dramatic reductions in runtime, the training process exhibited catastrophic degradation in convergence dynamics. This can be attributed to three interacting effects:

- *Statistical Inefficiency***:** At 48 ranks, each rank processes a very small dataset shard (≈ 1k samples), reducing gradient diversity per update and leading to stagnation.

- *Synchronization Overheads*: Frequent global gradient averaging across 48 processes amplifies communication costs, diluting effective learning despite reduced compute time per step.

- *Diminished Workload per Rank*: The fixed dataset (50k samples) cannot sustain high process counts under strong scaling, causing the training regime to fall into a regime where communication overhead and insufficient data per process jointly dominate, limiting convergence regardless

of raw speedup.

*Note*:

*The final loss values (≈ 2.3026) correspond to the entropy of a uniform random classifier across 10 classes. In deep learning terms, this indicates that the network predictions are essentially indistinguishable from random guessing. Unlike (Cases 1 and 2), where training and test loss tracked closely and convergence was evident, here both losses plateau at the random baseline, confirming that large-scale strong scaling under dataset-limited conditions prevents the model from learning effectively.*

- *Execution Time and Speedup:*

The strong scaling experiments clearly demonstrate the trade-off between execution time and effective learning. With (np = 2) (Case 1), the baseline configuration required (5003 sec) to complete 10 epochs, establishing the reference point ($S_2 = 1.0$, $E_2 = 100$ %). Doubling the nodes to (np = 4) (Case 2) reduced wall-clock time to (2535.6 sec), achieving nearly (2×) speedup ($S_4 \approx 1.97$) with very high efficiency ($E_4 \approx 99$ %). At np = 8, the speedup reached ($S_8 \approx 2.04$) and efficiency remained moderate (≈ 51 %), showing that the cluster scaled effectively up to this level. At full cluster scale (np = 48), wall-clock time collapsed to (432.1 sec), corresponding to an impressive raw speedup of (11.6×) relative to baseline “Table 2”, "Figure 8", "Figure 11".

- *Learning Performance: Accuracy and Loss*

While runtime scaled favorably, the learning performance of SqueezeNet degraded significantly as the number of processes increased. In the baseline configuration (np = 2), the model achieved (58.7 %) training accuracy and (61.6 %) test accuracy after 10 epochs, demonstrating that SqueezeNet can converge successfully under ARM-only execution. At np = 4, training accuracy dropped to (43.5 %) and test accuracy to (46.6 %), showing the early impact of reduced shard sizes and increased synchronization. By np = 8, accuracy fell further (≈ 32 % training and 35.6 % test), marking the point where statistical inefficiency begins to dominate, even though parallel efficiency remained moderate (≈ 51 %) “Table 2”, "Figure 10", "Figure 12".

At larger scales (np ≥ 16), convergence effectively collapsed. Both training and test accuracies stagnated near (10 %), with losses plateauing at (2.3026) — equivalent to random guessing across 10 classes. This collapse is not primarily due to poor parallel efficiency (which remained 40–50 % at these scales), but rather due to the extremely limited number of training images per rank. With only ~1k images per process at np = 48, the gradient signal was insufficient for learning, and global synchronization merely propagated noise. Such effects are consistent with prior findings that very small batch sizes introduce excessive gradient noise and destabilize convergence [10].

In short, the experiments reveal a scaling ceiling for distributed CNN training on edge-class ARM clusters. While modest scaling (np ≤ 8) balances throughput and learning quality, pushing to higher degrees of parallelism results in statistical underfitting: fast training with little or no useful convergence. This highlights that the bottleneck is not only communication overhead but also the intrinsic data-per-rank limitation in small-scale deep learning workloads on constrained hardware.

### *3.4.1 Scaling Summary*

The strong scaling experiments of *SqueezeNet* on the 24-node Raspberry Pi 4B cluster highlight the inherent trade-offs between execution speed and statistical learning efficiency.

From a runtime perspective, scaling was effective: wall-clock time for 10 epochs dropped from (5003 sec) at np = 2 (Case 1) to (2536 sec) at np = 4 (Case 2), and further down to just (432 sec) at full scale (np = 48). This corresponds to a raw speedup of (11.6×) compared to baseline, demonstrating that parallelization can indeed accelerate training throughput on embedded hardware. Parallel efficiency, defined relative to a baseline of np = 2 processes, remained high in the early regime (≈ 99 % at np = 4 and ≈ 51 % at np = 8), showing that the cluster scales well up to moderate sizes. However, beyond np = 8 the efficiency gradually declined, stabilizing around 40–50 % for larger process counts. This indicates that while communication and synchronization overheads do grow with scale [12], the more critical factor is statistical inefficiency: the workload per rank becomes too small, leaving each process with insufficient training data per epoch (a statistical bottleneck [11]). The experimental findings of this study on ARM-based edge clusters are consistent with the statistical bottlenecks described by Shallue et al. (2019) [11], where excessive parallelism leads to convergence collapse due to insufficient data per rank.

In terms of learning, performance degraded even more severely. The baseline run achieved (58.7 %) training accuracy and (61.6 %) test accuracy, proving that *SqueezeNet* can converge under ARM-only execution. At (*np = 4)*, accuracy dropped to (43.5 %) train and (46.6 %) test reflecting reduced learning capacity primarily due to smaller shard sizes (statistical bottleneck), with synchronization costs becoming secondary factors at larger scales. At full scale (*np = 48*), convergence collapsed entirely: mean training accuracy stagnated at (10.1 %) and both training and test loss plateaued at (2.3026), the entropy baseline of random guessing across 10 classes.

Taken together, these results demonstrate that while strong scaling improves execution time, it undermines statistical efficiency and generalization when pushed beyond dataset-limited thresholds. For lightweight CNNs such as SqueezeNet, distributed data-parallel training on edge clusters therefore exhibits a scaling ceiling: modest node counts (np ≤ 8) can balance throughput and accuracy, but aggressive scaling to dozens of nodes leads to rapid degradation of convergence, driven by the interplay of small per-rank batch sizes, synchronization overhead, and limited gradient signal.

### *3.4.2 Practical Implications for Edge AI*

The findings from this study underline a critical reality for edge-scale distributed deep learning. Raspberry Pi clusters, while inexpensive, energy-efficient, and highly customizable, face a sharp trade-off between scalability and learning performance when training compact CNNs such as SqueezeNet. For practical edge AI deployments — such as autonomous sensor networks, on-device vision analytics, or distributed IoT gateways — these results suggest that training should be confined to small-to-moderate node counts, where accuracy remains reliable and wall-clock time is still acceptable. At larger scales, the diminishing returns of parallel efficiency and the collapse of convergence observed at np = 48 imply that such clusters are better suited for inference and lightweight retraining rather than full-scale distributed training.

Nevertheless, the empirical profiling carried out here (e.g., identification of batch size = 24 as the throughput/memory

optimum) demonstrates that hardware-aware tuning is essential: even within resource-constrained settings, careful optimization can yield stable convergence without overfitting. Thus, Raspberry Pi–based *SqueezeNet* clusters can play a meaningful role as testbeds for edge AI research, enabling low-cost exploration of distributed learning paradigms, energy-performance trade-offs, and algorithmic strategies (e.g., adaptive synchronization, hybrid training) that could later transfer to industrial IoT or mission-critical edge applications.

Ultimately, this work shows that while such embedded clusters will not replace GPU datacenters for large-scale deep learning, they occupy a unique niche: providing accessible, reproducible, and hardware-constrained platforms where the challenges of distributed training at the network's edge can be studied under realistic conditions — conditions that mirror the limitations of real-world deployments in remote, mobile, or power-sensitive environments.

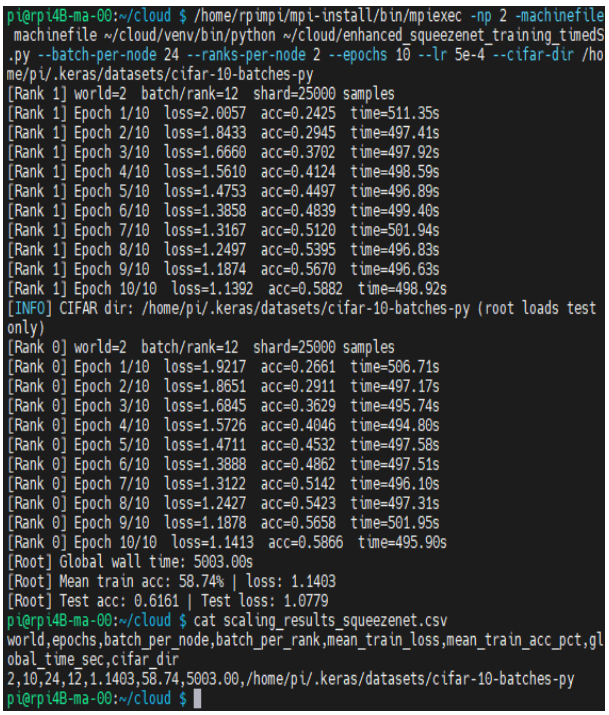


**Figure 6: SqueezeNet training on a single Raspberry Pi 4B with two MPI processes (np = 2).**

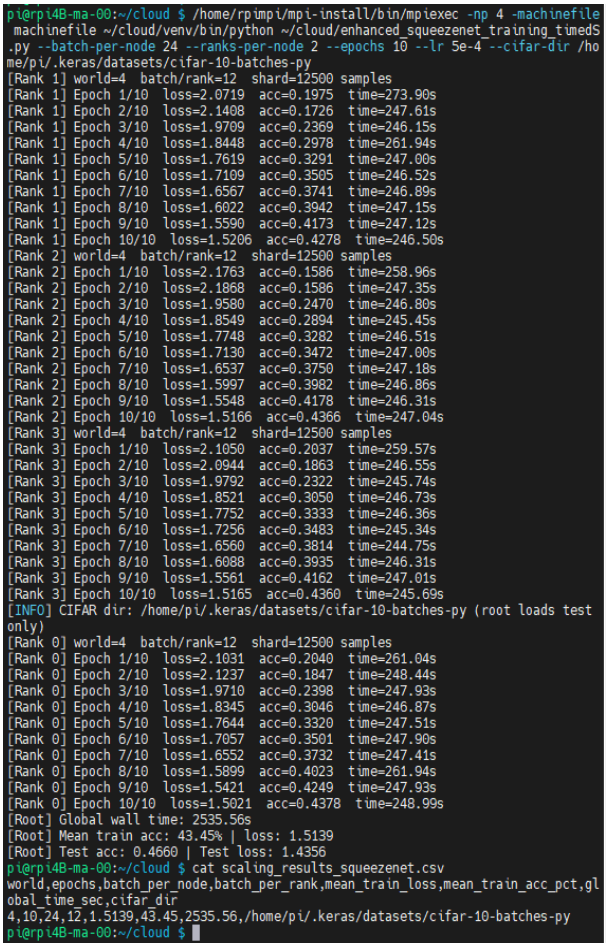


**Figure 7: SqueezeNet training on two Raspberry Pi 4B nodes with four MPI processes (np = 4).**

**Table 2. SqueezeNet CNN Model Training results: Strong Scaling Methodology**

| RPi's | MPI Processes (np) | Epoch | Test acc (final) (%) | Test loss (final) | Mean Train Loss (unitless) (≈) | Mean Train Accuracy (%) | Total (wall) Training Time (slowest rank) (Mean) (sec) | Speedup ($S_n$) | Efficiency ($E_n$) (%) |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 2 | 10 | 61.61% | 1.0779 | 1.1403 | 58.74% | 5003.00 | 1 | 100% (baseline) |
| 2 | 4 | 10 | 46.60% | 1.4356 | 1.5139 | 43.45% | 2535.56 | 1.97313414 | 98.66% |
| 4 | 8 | 10 | 35.61% | 1.7313 | 1.7995 | 32.06% | 2455.94 | 2.037101884 | 50.93% |
| 8 | 16 | 10 | 10.00% | 2.3026 | 2.3026 | 10.01% | 1156.48 | 4.326058384 | 54.08% |
| 12 | 24 | 10 | 10.00% | 2.3026 | 2.3026 | 10.05% | 851.04 | 5.878689603 | 48.99% |
| 16 | 32 | 10 | 10.00% | 2.3026 | 2.3026 | 10.00% | 699.52 | 7.152047118 | 44.70% |
| 20 | 40 | 10 | 10.00% | 2.3026 | 2.3026 | 10.16% | 627.32 | 7.975196072 | 39.88% |
| 24 | 48 | 10 | 10.00% | 2.3026 | 2.3026 | 10.10% | 432.11 | 11.5780704 | 48.24% |

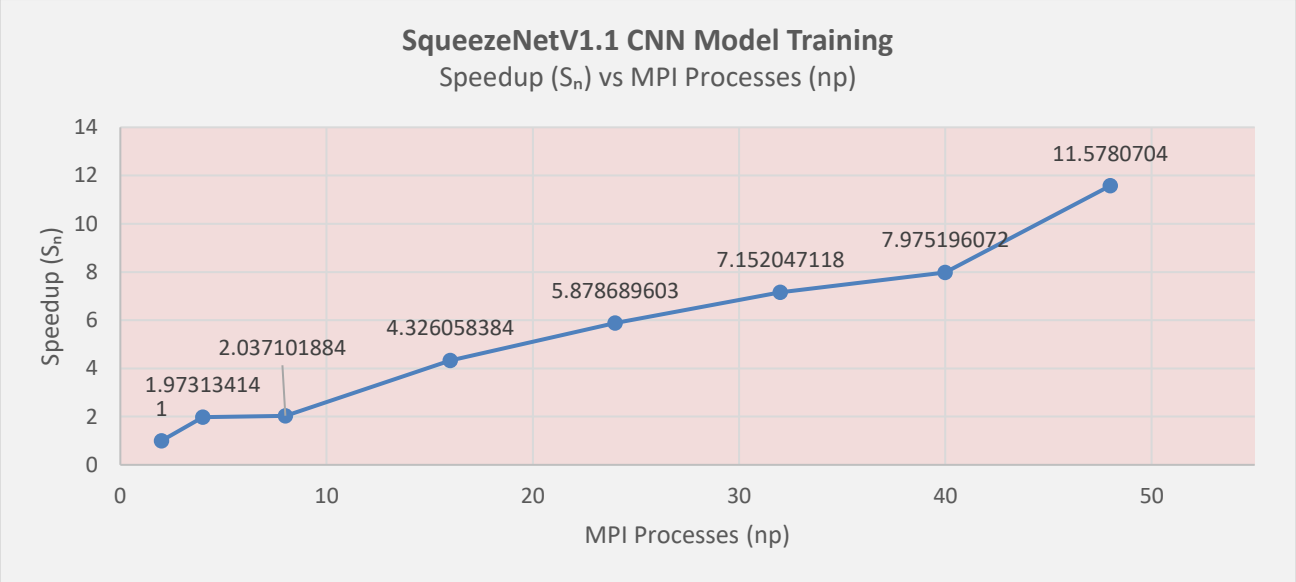


**Figure 8: SqueezeNet CNN Model Training: Speedup ($S_n$) vs MPI Processes (np)**

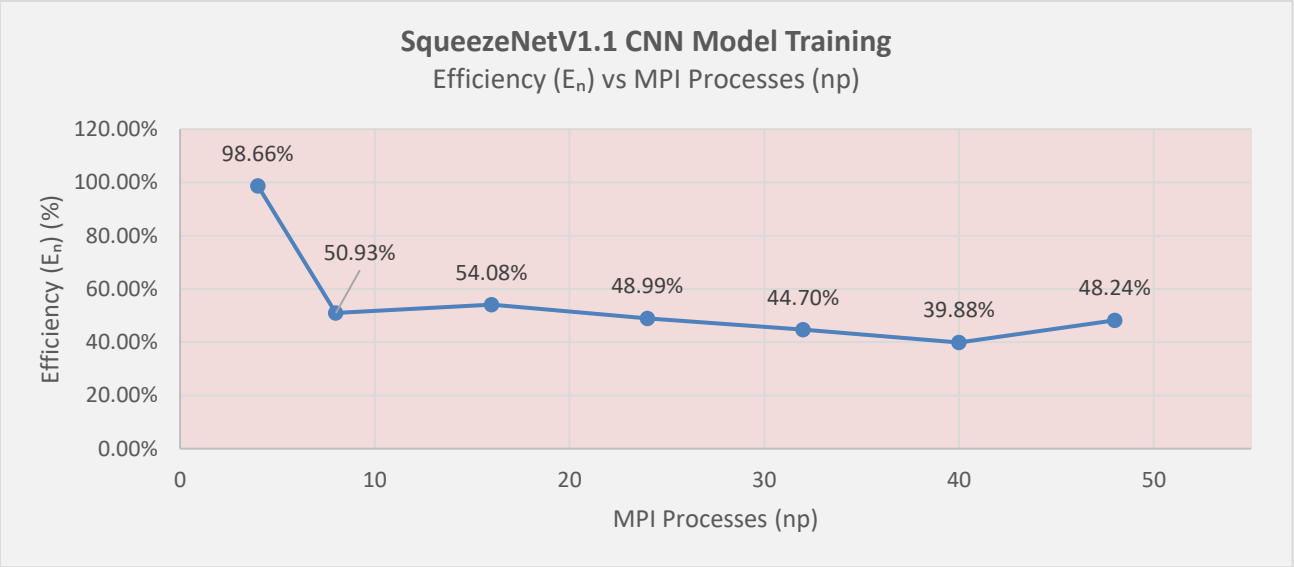


**Figure 9: SqueezeNet CNN Model Training: Efficiency ($E_n$) vs MPI Processes (np)**

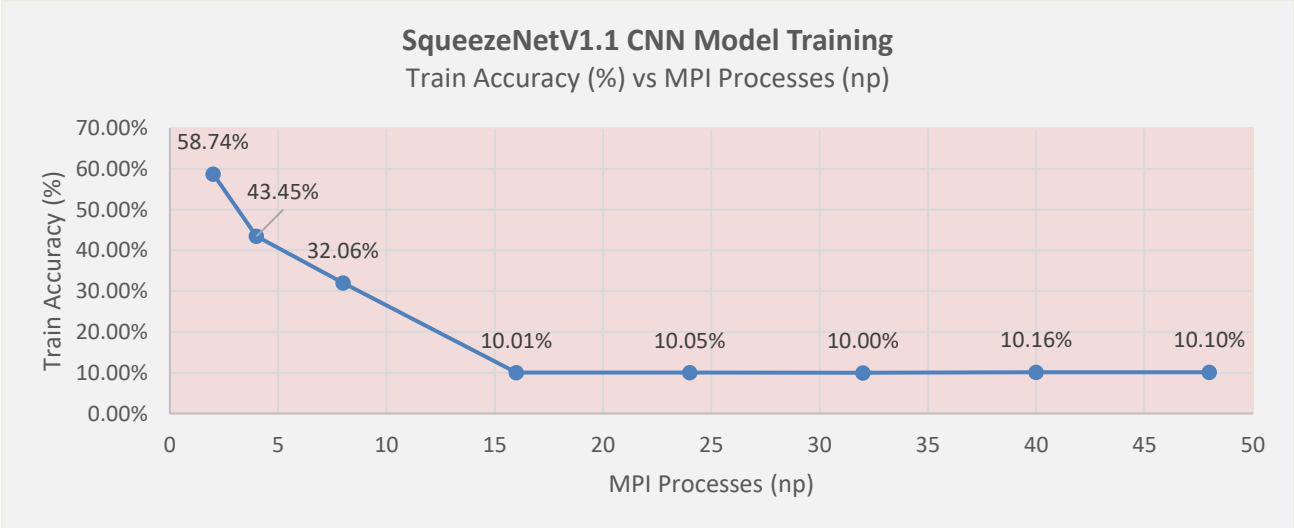


**Figure 10: SqueezeNet CNN Model Training: Train Accuracy (%) vs MPI Processes (np)**

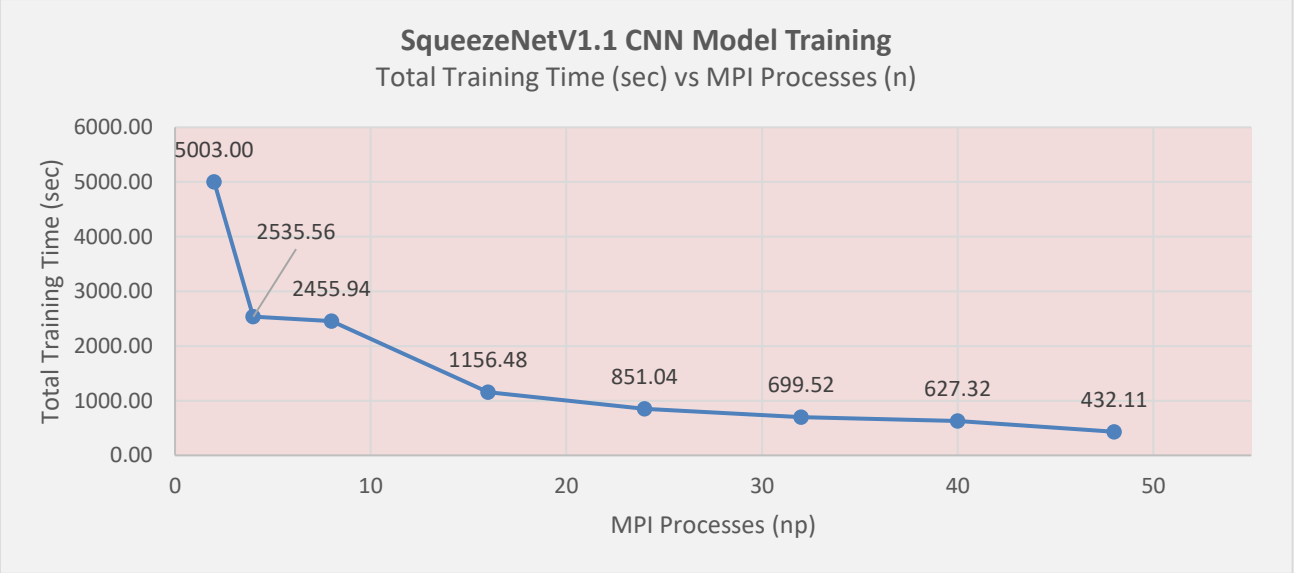


**Figure 11: SqueezeNet CNN Model Training: Total Training Time (sec) vs MPI Processes (n)**

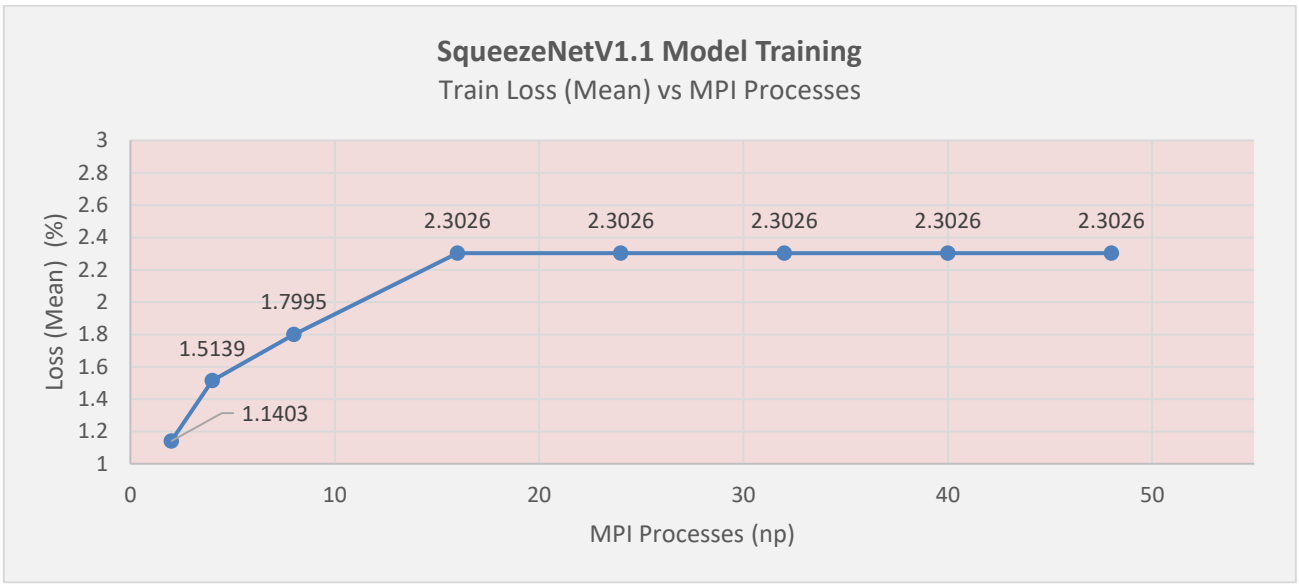


**Figure 12: SqueezeNet CNN Model Training: Train Loss (Mean) vs MPI Processes**

## 4. FUTURE WORK

The limitations observed in the distributed training of SqueezeNet - particularly in convergence stability and scalability across high-rank ARM clusters - motivate a deeper investigation into alternative learning paradigms that are inherently designed for energy efficiency and sparse computation.

In this context, Spiking Neural Networks (SNNs) represent a biologically-inspired and event-driven approach to deep learning that aligns naturally with edge constraints. As a continuation of this work, there is a plan for the authors to explore Distributed SNN Training on ARM-Based Edge Clusters using MPI, evaluating how spike-based models behave under partitioned data and low-bandwidth communication regimes. Building on the execution pipeline developed for SqueezeNet, the authors have intention to implement SNNs that leverage Loihi- and SpiNNaker-inspired simulation frameworks, adapting their temporal and sparse encoding schemes to fit CPU-bound environments.

This line of research not only extends the architectural diversity of models studied under MPI-distributed schemes, but also serves as a potential path toward ultra-low-power edge intelligence, where SNNs may outperform traditional CNNs like SqueezeNet in both computational cost and biological plausibility.

## 5. CONCLUSION

This study investigated the distributed training performance of the SqueezeNet CNN model using MPI and mpi4py on a 24-node ARM-based edge cluster. Experimental results across configurations from RPi-1_mpi-2 to RPi-24_mpi-48 revealed that although SqueezeNet maintains low computational complexity, its convergence behavior under data-parallel MPI execution is inconsistent beyond moderate scale. Notably, training accuracy plateaued and even degraded at higher process counts, indicating limited benefit from aggressive parallelism. The results empirically confirm on ARM-based edge clusters the statistical bottlenecks previously reported by Shallue et al. (2019), [11] in large-scale GPU/CPU training environments, underscoring that the convergence collapse we

observed is not hardware-specific but a fundamental limitation of excessive data parallelism when per-rank shard sizes fall below critical thresholds.

In contrast to structurally heavier models like MobileNet, SqueezeNet does not linearly benefit from MPI-based scaling in low-power, bandwidth-constrained environments. These findings emphasize that lightweight architectures, while computationally efficient, may require careful tuning — such as batch-size adjustment, hybrid pipelining, or adaptive gradient aggregation — to sustain effective distributed learning. Future work will extend this investigation toward distributed Spiking Neural Networks (SNNs) using MPI, aiming to assess the viability of neuromorphic models and Loihi/SpiNNaker-inspired simulations on ARM clusters for scalable and biologically plausible edge intelligence.

A distinctive contribution of this work lies in the systematic derivation of optimal training hyperparameters through empirical profiling rather than reliance on recommended defaults. Single-node sweeps demonstrated that a batch size of 24 images per node maximizes throughput (≈ 19.9 img/s), maintains safe memory utilization (≈ 638 MB per rank), and fully saturates available CPU resources, while avoiding cache and bandwidth penalties observed at larger batches. This hardware-aware tuning procedure establishes a reproducible empirical baseline for distributed training on ARM clusters, underscoring the necessity of hyperparameter selection guided by platform constraints rather than convention.

In summary, this work positions Raspberry Pi–based *SqueezeNet* clusters not as replacements for datacenter-scale training, but as accessible and realistic testbeds for studying distributed deep learning under edge constraints. Such platforms directly mirror the computational and bandwidth limitations of real-world IoT, robotics, and environmental monitoring systems, and thus provide practical insights into how compact CNNs — and potentially SNNs — can be trained and deployed in the next generation of decentralized, low-power edge intelligence.

## 6. ACKNOWLEDGMENTS

My sincere gratitude to Assistant Professor Ioannis S. Barbounakis for his precious guidelines, knowledge and contribution for the completion of this study.

## 7. REFERENCES

[1] Shi, W., Cao, J., Zhang, Q., Li, Y., & Xu, L. (2016). *Edge computing: Vision and challenges*. IEEE Internet of Things Journal, 3(5), 637–646. https://doi.org/10.1109/JIOT.2016.2579198.

[2] Li, S., Xu, L. D., & Zhao, S. (2018). 5G Internet of Things: A survey. *Journal of Industrial Information Integration*,*10*,1-9 https://doi.org/10.1016/j.jii.2018.01.005

[3] Sze, V., Chen, Y. H., Yang, T. J., & Emer, J. S. (2017). Efficient processing of deep neural networks: A tutorial and survey. *Proceedings of the IEEE*, 105(12), 2295–2329. https://doi.org/10.1109/JPROC.2017.2761740

[4] Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., … & Adam, H. (2017). MobileNets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint* arXiv:1704.04861. https://arxiv.org/abs/1704.04861

[5] Iandola, F. N., Han, S., Moskewicz, M. W., Ashraf, K., Dally, W. J., & Keutzer, K. (2016). SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <0.5MB model size. *arXiv Preprint*. https://doi.org/10.48550/arXiv.1602.07360

[6] Li, H., Kadav, A., Durdanovic, I., Samet, H., & Graf, H. P. (2017). Pruning filters for efficient convnets. *International Conference on Learning Representations (ICLR)*. https://arxiv.org/abs/1608.08710

[7] Ramesh, S., & Chakrabarty, K. (2021). Challenges and opportunities in training deep neural networks on edge devices. *ACM Transactions on Embedded Computing Systems (TECS)*, 20(5s), 1–26. https://doi.org/10.1145/3477084

[8] Raspberry Pi 4 Model B. [Online]. Available: raspberrypi.com/products/raspberry-pi-4-model-b/.

[9] Raspberry Pi 4 Model B specifications. [Online]. Available: https://magpi.raspberrypi.com/articles/raspberry-pi-4-specs-benchmarks.

[10] Masters, D., & Luschi, C. (2018). Revisiting small batch training for deep neural networks. *arXiv preprint* arXiv:1804.07612. https://arxiv.org/abs/1804.07612

[11] Shallue, C. J., Lee, J., Antognini, J., Sohl-Dickstein, J., Frostig, R., & Dahl, G. E. (2019). Measuring the effects of data parallelism on neural network training. *Journal of* Machine Learning Research, 20(112), 1–49. http://jmlr.org/papers/v20/18-789.html

[12] Ben-Nun, T., & Hoefler, T. (2019). Demystifying parallel and distributed deep learning: An in-depth concurrency analysis. *ACM Computing Surveys*, 52(4), 1–43. https://doi.org/10.1145/3320060